

APPLICATION FOR UNITED STATES PATENT

SYSTEM AND METHOD FOR SECURE AND VERIFIED SHARING OF RESOURCES IN A PEER-TO-PEER NETWORK ENVIRONMENT

INVENTORS: **Victor Kouznetsov**
20287 SW Tremont Way
Aloha, OR 97007
A Citizen of Russia

Charles L. Vigue
PO Box 1736
Lapine, OR 97007
A Citizen of United States

Martin Fallenstedt
9045 S.W. 182nd Avenue
Beaverton, OR 97007
A Citizen of United States

Daniel Melchione
10380 SW 152nd Terrace
Beaverton, OR 97739
A Citizen of United States

ASSIGNEE: **Networks Associates Technology, Inc.**
3965 Freedom Circle
Santa Clara, CA 95054
A Delaware Corporation

ENTITY: **Large**

Jung-hua Kuo
Attorney at Law
P.O. Box 3275
Los Altos, CA 94024
Tel: (650) 988-8070
Fax: (650) 988-8090

SYSTEM AND METHOD FOR SECURE AND VERIFIED SHARING OF RESOURCES IN A PEER-TO-PEER NETWORK ENVIRONMENT

Victor Kouznetsov
Charles L. Vigue
Martin Fallenstedt
Daniel Melchione

CROSS REFERENCE TO RELATED APPLICATIONS

This application claims the priority benefit of U.S. Provisional Patent Application No. 60/282,333, entitled "System and Method for Efficient Use of Bandwidth and Resources in a Peer-to-Peer Network Environment" and filed April 6, 2001 and U.S. Provisional Patent Application No. 60/298,681, entitled "System and Method for Efficient Updating of Virus Protection Software and Other Efficient Uses of Bandwidth and Resources in a Peer-to-Peer Network Environment" and filed June 15, 2001, both of which are incorporated herein by reference in their entireties.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates generally to a system and method for efficient use of bandwidth and resources in a peer-to-peer network environment. More specifically, a system and method for secure and verified sharing of resources in a peer-to-peer network environment to facilitate efficient use of bandwidth are disclosed.

2. Description of Related Art

Conventionally, to obtain anti-virus product updates and/or signature files, computers rely on a pull approach in which each client or server computer must retrieve the updated anti-virus file directly from a source via the Internet. For a computer network, a network administrator may allow anti-virus signature files to become out of date because there are simply too many clients on the network for effective management. Alternatively, the network administrator may schedule the clients to automatically pull the updated anti-virus file from the Internet when each client logs onto the computer. However, such an approach can result in a bandwidth crunch such as in the early morning work hours when most users log onto their computers.

Connections to the Internet from within an organization, particularly from a small to medium sized organization, may be relatively slow. For example, a small to medium sized business may share a single cable or DSL modem, a 56K modem, or an ISDN line. In contrast, in a typical work group interconnected via a LAN, connections on the LAN are generally much faster, the typical LAN being 100/TX (100 Mbps). Peer-to-peer networks thus partially address the need for efficient use of bandwidth and resources in a computer network.

However, conventionally, peer-to-peer distribution is not secure and is particularly susceptible to attacks since a desired resource or a file on the network may be compromised as a file distributor does not have control of the peer servers that are passing around copies of the file on the network. This is of particular concern for files that are actual executable binaries and/or for files containing potentially critical data such as anti-virus product updates and signature files. Furthermore, for product updates such

as anti-virus product updates, a requesting peer may not be able to determine whether the product files on other peer computers are up-to-date or outdated.

Thus, it is desirable to provide a system and method for secure and verified sharing of resources in a peer-to-peer network environment to facilitate efficient use of bandwidth to ensure that the shared file has not been compromised and/or is up-to-date. Such secure and verified sharing of resources over a peer-to-peer network ideally expands network resource sharing into broader applications such as enterprise software management and services.

SUMMARY OF THE INVENTION

A system and method for secure and verified sharing of resources in a peer-to-peer network environment to facilitate efficient use of bandwidth are disclosed. The peering service system and method facilitate in spreading load amongst peers in a distributed network interconnected via a LAN in a smooth, secure and scalable way. The peering service provides security mechanisms to ensure that the shared file has not been compromised and/or is up-to-date. Preferably, the service-enabled application, such as an updating application, implements digital signature security to verify the integrity of the retrieved data in order to ensure that the resource, such as an update, has not been compromised. The service-enabled application preferably also implements version information security measure to ensure that the resource is an up-to-date or otherwise correct version.

It should be appreciated that the present invention can be implemented in numerous ways, including as a process, an apparatus, a system, a device, a method, or a

computer readable medium such as a computer readable storage medium or a computer network wherein program instructions are sent over optical or electronic communication lines. Several inventive embodiments of the present invention are described below.

According to a preferred embodiment, a method for securely sharing resources over a peer-to-peer network generally comprises broadcasting a request by a requesting peer for a resource over the peer-to-peer network where the resource is identified with a resource version identifier, receiving a response from a responding peer on the peer-to-peer network indicating that the responding peer has the requested resource, retrieving the requested resource from the responding peer, and verifying the retrieved resource by ensuring the retrieved resource contains the version identifier embedded therein. Preferably, the verifying also includes verifying a digital signature, such as a 1024-bit VeriSign digital certificate, of the retrieved resource to ensure integrity of the retrieved resource.

The method may further include retrieving a catalog containing a listing of resources, comparing the resource listing with resources installed at the requesting peer to determine which resources are to be requested over the peer-to-peer network, and requesting each such resource in a separate transaction such that each such resource may be retrieved from a same or different responding peer.

According to another preferred embodiment, a product updating service for automatic and secure updating of a product installed at a node of a peer-to-peer network generally comprises automatically downloading a catalog containing a current listing of resources for the product at a predetermined time, comparing the listing with resources installed at the node to determine which resources are to be requested, requesting each

such resource in a separate transaction over the peer-to-peer network, retrieving each resource from a peer in the peer-to-peer network or the Internet, and verifying each retrieved resource by ensuring the retrieved resource contains a version identifier embedded therein.

5 According to yet another preferred embodiment, a method for providing secure updating of a software product generally comprises providing for retrieval over the Internet a catalog containing a current listing of resources for the product and providing for retrieval over the Internet the resources for the product, each resource being identified with a resource version identifier and contains the resource version identifier embedded
10 therein and/or contains a digital signatures such as a 1024-bit VeriSign digital certificate.

These and other features and advantages of the present invention will be presented in more detail in the following detailed description and the accompanying figures which illustrate by way of example the principles of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

15 The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

FIG. 1 is a block diagram of an exemplary computer network suitable for
20 implementing a peering service in a peer-to-peer network to facilitate efficient use of bandwidth and resources;

FIG. 2 is a block diagram illustrating an exemplary peering service system and method implemented at a node of the computer network of **FIG. 1**;

FIG. 3 is a state diagram illustrating states of a typical peering service server in processing a request from a peering client in the peer-to-peer network;

FIGS. 4A and **4B** are alternative state diagrams illustrating states of a typical peering service client in requesting a resource over the peer-to-peer network;

5 **FIG. 5** is a flowchart illustrating a typical process of a peering service server in processing a request from a peering client in the peer-to-peer network;

FIG. 6 is a flowchart illustrating a typical process of a peering service client in requesting a resource over the peer-to-peer network;

FIG. 7 is a flowchart illustrating a preferred embodiment of the retrieve step of

10 **FIG. 6** in more detail;

FIG. 8 is a flowchart illustrating an exemplary secure product updating process implemented by a service-enabled product updating application over a peer-to-peer network;

15 **FIG. 9** illustrates an example of a computer system that can be utilized with the various embodiments of method and processing described herein; and

FIG. 10 illustrates a system block diagram of the computer system of **FIG. 9**.

DESCRIPTION OF SPECIFIC EMBODIMENTS

20 A system and method for secure and verified sharing of resources in a peer-to-peer network environment to facilitate efficient use of bandwidth are disclosed. The peering service facilitates in spreading load amongst peers in a distributed network interconnected via a LAN in a smooth, secure and scalable way. A service or an application that is service-enabled may minimize or reduce the usage of, for example,

Internet bandwidth by attempting to locate a local aliased copy of a requested resource residing within the peer-to-peer network. If a local aliased copy of the requested resource is located, the requesting computer may obtain the requested resources locally. Once the requesting computer obtains a copy of the requested resource, whether locally or
5 remotely, the requesting computer may itself become a server for the aliased copy for subsequent requests for the resource.

The following description is presented to enable any person skilled in the art to make and use the invention. Descriptions of specific embodiments and applications are provided only as examples and various modifications will be readily apparent to those
10 skilled in the art. The general principles defined herein may be applied to other embodiments and applications without departing from the spirit and scope of the invention. Thus, the present invention is to be accorded the widest scope encompassing numerous alternatives, modifications and equivalents consistent with the principles and features disclosed herein. For purpose of clarity, details relating to technical material that
15 is known in the technical fields related to the invention have not been described in detail so as not to unnecessarily obscure the present invention.

FIG. 1 is a block diagram of an exemplary computer network 100 suitable for implementing the peering service in a peer-to-peer network to facilitate efficient use of bandwidth and resources as described herein. In particular, the computer network 100
20 comprises nodes, computers, or workstations 104A-E interconnected via a LAN 102. It is to be understood that the LAN 102 may be implemented using any suitable network mechanism including wire and wireless. In the exemplary computer network 100, only two of the nodes 104D and 104E have access to the Internet.

FIG. 2 is a block diagram illustrating an exemplary peering service system and method implemented at a node of the computer network of **FIG. 1**. As shown, each node 104 provides the functionality of both a server 106 and a client 110. The peering service system utilizes a port, such as port 1967, for transmitting directed or broadcast messages to peers on the network. The server preferably includes an embedded HTTP server 108, typically a micro HTTP server. The HTTP server 108 allows aliased URLs to be accessed by other peers on the network. The HTTP server 108 preferably uses an obscure port such as port 6515 and is preferably restricted to operations required to facilitate distribution of, for example, cached files and uploading of data or requests.

Typically, each node runs both the server and the client. However, each node may run only the client or the server. The peering system and method are preferably implemented as a peering service application (“service” or “service-enabled application”) or daemon process. It is noted that a service-enabled application need not be a service application. For example, a service-enabled application may also refer to a service-aware application that fires up, communicates with the service and then shuts down interactively.

The peering system preferably provides a linkable client API library 112 to facilitate communication between the peering service and any service-enabled applications. In one preferred embodiment, the peering system may export the client API library 112 to any service-enabled application such that the service-enabled application may utilize the peering service to discover any type of resource that can be identified with a URL or URI, for example. Alternatively, a given application and the peering

service may be tightly coupled so as to eliminate the need for the linkable client API library.

FIG. 3 is a state diagram illustrating states 120 of a typical peering service server in processing a given request from a peering client in the peer-to-peer network. Initially, the service server is in an idle state 122 while listening on a designated port for a broadcast request message from a peer client on the network. When the service server receives a broadcast request message such as an “I need” packet from a peering client on the network, the service server transitions to a locating local aliased copy state 124. In particular, the service server refers to its list of local aliased copies to determine if the service server has a local copy of the requested resource or item identified by, for example, an URL/URI. If the service server determines that it does not have a local copy of the requested resource, then the service server returns to the server idle state 122.

Alternatively, if the service server determines that it has a local copy, the service server preferably waits a randomly generated delay response time period at stage 126. The service server may generate a random number, such as between 0 and 2000, which the service server utilizes as the length of time it waits before responding. In one preferred embodiment, the random number is the number of milliseconds the service server waits before replying to the request. While the service server awaits expiry of the randomly generated delay response time period, the service server listens for a broadcast “I found” packet from the requesting client corresponding to the received request packet. It is noted that regardless of the state of the service server for a given peer request, the service server listens for new requests such as “I need” packets. The broadcast “I found” packet from the requesting client indicates that the requesting client has found the

requested resource. If the service server receives an "I found" packet from the requesting client before expiry of the delay response time period, the service server transitions to state 128 to cancel the response to the request and returns to server idle state 122.

Alternatively, if no "I found" packet is received prior to the expiration of the delay response time period, the service server transitions to state 130 to transmit an "I have" packet directly to the requesting peer client. The "I have" packet preferably contains a local alias for the requested object on the service server which the requesting peer can access via the HTTP server of the of the service server. Although not preferred, the service server may alternatively broadcast the "I have" packet over the network rather than transmitting it directly to the requesting client. The service server then returns to the server idle state 122.

As is evident, the randomly generated delay response time period allows multiple peer servers to share loads in an orderly fashion. In particular, randomizing the delay response time period ensures that any given node would not automatically become the default server to a large portion of the peers and eliminates any need for the service server to exercise preferences as to which service clients the service server will supply the requested item. In other words, the random wait time before responding to a request ensures that any one machine does not become an overloaded server of the item or update to the rest of the network. In addition, as a given item is propagated through the network to peers on the network, the load on any one node is likely further reduced. Thus, the system impact on a given service server as it supplies the requested item to service clients can be relatively minimal.

However, it is to be understood that a situation in which multiple service servers each transmitting an “I have” packet in response to a given request packet may occur. For example, a first service server may transmit an “I have” packet upon expiry of its delay response time period. The “I found” packet transmitted or to be transmitted by the requesting peer corresponding to the first “I have” packet may not arrive at the second service server prior to the expiry of its delay response time period, causing the second service server to transmit an “I have” upon expiry of its delay response time period. In such a situation where the requesting client receives multiple “I have” packets from multiple service servers, the requesting client may simply process the first “I have” response and ignore any subsequent “I have” packets it may receive.

FIGS. 4A and 4B are alternative state diagrams illustrating states 140, 140A of a typical peering service client in making a given request for a resource over the peer-to-peer network. Referring to **FIG. 4A**, initially, the service client is in an idle state 142. When a service client needs a desired resource, such as an Internet resource, the service client generates and broadcasts an “I need” packet over the peer-to-peer network. For example, the “I need” request may specify an URL (e.g., <http://something.tld/someother/thing/here>), a protocol (e.g., HTTP protocol), a desired operation (e.g., get operation), and that the requesting peer only wants cached objects.

After broadcasting the “I need” request, the service client transitions to a waiting for response state 144 in which the service client awaits a maximum delay response time period plus a transmission time period for a response from any of the service servers. In the example above where the randomly generated delay response time period ranges

between 0 and 2000 milliseconds, the client response waiting time period would be, for example, 2200 milliseconds to allow for a 200 millisecond transmission time period.

If an "I have" response is received from a service server during the client response waiting time, then the service client transitions to state 146 and generates and broadcasts an "I found" message over the network to inform all other peers that the desired resource or item has been found. The service client then transitions to the requested item found state 158. The service-enabled application requesting the item then retrieves the requested item from the responding service server at the location within the network as specified in the received "I have" packet. Generally, the service-enabled application retrieves the requested item through the local HTTP server using, for example, port 6515. Once the service-enabled application successfully retrieves the requested item, the service client informs the service server running on the same machine that a local copy of the resource now exists. The service client then returns to the idle state 142.

Alternatively, if no response is received during the client response waiting time, the service client times out and transitions to state 150 to retrieve the requested item itself such as via the Internet. Once the retrieval is complete, the service client transitions to found item state 158 in which the service client informs the service server running on the same computer or at the same node that a local copy of the resource now exists. The service client then returns to client idle state 142. As is evident, regardless of whether the service client received an "I have" packet from a service server on the network, the client machine can itself become a service server for the requested resource after successful completion of its request.

FIG. 4B illustrates the 140A states of a typical service in a preferred alternative embodiment particularly suitable for applications that include downloading of files.

States 140A includes the states as shown and described with reference to **FIG. 4A** plus additional states for dealing with currently in-progress downloads of the requested item by other peers. These additional states allow a peer node to complete downloading the requested resource and then distribute it immediately and automatically upon download completion to the requesting service client.

In particular, instead of directly transitioning to state 150 to retrieve the requested item itself after the service client times out the request, the service client transitions to “wait for download?” state 144 in which the service client determines whether it can or will wait for completion of any in-progress download of the requested item by another peer. If not, then the service client transitions to state 150 to retrieve the requested item itself and continues with state transitions similar to that described above with reference to **FIG. 4A**.

If the service client determines that it can or will wait for the completion of any in-progress download, the service client transitions to “any in-progress downloads?” state 152. If there are no such in-progress downloads of the requested item, then the service client transitions to state 150 to retrieve the requested item itself and continues with state transitions similar to that described above with reference to **FIG. 4A**.

Alternatively, if there is at least one in-progress download of the requested item, then the service client transitions to state 154 in which it generates and broadcasts an “I found” message. The service client then transitions to state 156 to await completion of the in-progress download of the requested item. Upon completion of the in-progress

download of the requested item, the service client transitions to the requested item found state 158. The service client retrieves the requested item from the local location within the network. After successful completion of its request, the service client will inform the service server running on the same machine that a local copy of the resource now exists.

5 The service client then returns to the idle state 142.

As is evident, in order for the service client to determine if there are any in-progress downloads in state 152, a service client that is downloading a file for a service-enabled application preferably broadcasts a “downloading” message and/or directly responds to the client server of a broadcast “I need” request with a “I am downloading” rather than an “I have” response message. In one preferred embodiment, the service client may set a downloading flag for the corresponding file to true.

In addition, the service-enabled application preferably transmits periodic progress packets to any node that is waiting for the resource being downloaded such that those nodes may interactively display download progress information to end users at state 156. Alternatively, the service-enabled application may broadcast such periodic download progress packets over the network. Thus, a node in the retrieve item state 150 preferably periodically transmits a “downloading” message that includes progress information.

Service Functionality and Service Packet Format

20 One functionality provided by the peering service is that of a central clearing house for formatting, sending, receiving and decoding service packets, such as for “I need,” “I found,” and “I have” packets. In other words, the peering service manages the peer-to-peer communication process for obtaining requested items. The specific

functionality invoked by a given service packet itself is generally dependent on the specific service-enabled application.

The communication protocol used in broadcasts (e.g., “I need” and “I found” packets) and responses (e.g., “I have” packets) is typically TCP/IP. Each packet is typically approximately 200 bytes in size and contains the node ID of the sender as well as any other suitable information. Transfer of the requested item from the service server to the service client is typically via HTTP.

The service packet format is preferably based upon the well-accepted and widely utilized XML format. For example, an XML service packet format may include a service identification and various key-value pairs, including those inserted by the service as well as those defined by the corresponding service-enabled application.

Various key-value pairs may be inserted by the peering service into each service packet. Examples of suitable key-value pairs include identification, type, and version key-value pairs. Specifically, an identification key-value pair identifies each request and responses corresponding to the request. In general, the identification value is unique on the originating node but need not be unique on the network as a whole. The range of values for the identification value may depend upon the number of bits assigned thereto. For example, 32 bits or four octets may be assigned to the identification value and thus the identification value would range from 0 to 2^{31} . With respect to the type key-value pair, the type value is typically either request, end-request, response, and/or any application-defined value. Any other suitable application-defined key-value pairs may also be included in the service packet.

An exemplary service packet may be:

```
<service type = "request" version = "1.0" ID = "1111" method = "get"  
href = "http://domain.com/whatever" acceptprotocol = "http"/>
```

5 **FIG. 5** is a flowchart illustrating a typical process 180 of a peering service server in processing a request from a peering client in the peer-to-peer network. At step 182, the service server is listening on a designated port for a broadcast request message from a peer client on the network. At step 184, the service server receives a broadcast request message on the designated port such as an "I need" packet from a peering client on the
10 network. At step 186, the service server determines if it has a local aliased copy of the requested item. In particular, the service server refers to its list of local aliased copies to determine if the service server has a local version of the requested resource or item, such as an URL/URI.

15 If the service server determines that it does not have a local copy of the requested resource, then the process 180 is complete. Alternatively, if the service server determines that it has a local copy, the service server preferably waits a randomly generated delay response time period while listening for a broadcast "I found" packet from the requesting client corresponding to the received request packet at step 188. As discussed above, the service server may generate a random number between 0 and 2000 as the length of time
20 in milliseconds it waits before responding. The broadcast "I found" packet from the requesting client indicates that the requesting client has found the requested resource.

It is noted that throughout the process 180, the service server is preferably continuously listening for any additional broadcast request messages and performs process 180 for each received broadcast request message as they are received.

If the service server receives an “I found” packet from the requesting client before expiry of the delay response time period, the service server cancels the response to the request at step 190 and the process 180 is complete. Alternatively, if no “I found” packet is received prior to the expiration of the delay response time period, the service server transmits an “I have” packet directly to the requesting peer client at step 192 and the server process 180 is complete. The “I have” packet preferably contains a local alias for the requested object on the service server.

FIG. 6 is a flowchart illustrating a typical process 200 of a peering service client in requesting a resource over the peer-to-peer network. At step 202, the service client generates and broadcasts an “I need” packet over the peer-to-peer network on a designated port. At step 204, the service awaits for a response from any of the service servers on the network for a period equal to a client response waiting time period, typically a maximum delay response time period plus a transmission time period, .

If an “I have” response is received from a service server during the client response waiting time, then the service client generates and broadcasts an “I found” message over the network at step 206. The service-enabled application requesting the item then retrieves the requested item from the responding service server at the location within the network as specified in the received “I have” packet at step 208. Once the service-enabled application successfully retrieves the requested item, the service client informs

the service server running on the same machine that a local copy of the resource now exists at step 210. The process 200 is then complete.

Alternatively, if no response is received during the client response waiting time, i.e., the service client times out, the service client determines if the service-enabled application can or will wait for completion of any in-progress download of the requested item by another peer at step 214. If not, the service client retrieves the requested item itself such as via the Internet at step 216 and then proceeds to step 210 to complete the process 200.

If the service client determines that it can or will wait for the completion of any in-progress download, the service client determines whether there are any in-progress downloads at step 220. If there are no such in-progress downloads of the requested item, the service client then proceeds to step 210 to complete the process 200.

If there is at least one in-progress download of the requested item, then the service client generates and broadcasts an "I found" message at step 222. The service client then awaits completion of the in-progress download of the requested item at step 224. For example, the service client may receive an "I have" or a "Download complete" message from the downloading peer.

Upon completion of the in-progress download of the requested item, the service client retrieves the requested item from the local location within the network at step 226. After successful completion of its request, the service client then proceeds to step 210 to complete the process 200. It is noted that steps 214 and 220-226 can be optional and preferably implemented for applications that include downloading of files

As is evident, in order for the service client to determine if there are any in-progress downloads at step 220, a service client that is downloading a file for a service-enabled application from outside of the network, e.g., from the Internet, notifies its peers on the network that a downloading process is in progress. For example, FIG. 7 is a flowchart illustrating a preferred embodiment of the retrieve step 216 in more detail.

As shown, the service client begins retrieving the requested item at step 216A. At step 216B, the service client may broadcast a “downloading” message and/or directly respond with a “I am downloading” response message to any client server that transmitted a broadcast “I need” request. In addition, the service client preferably also periodically transmits progress packets at step 216C either by broadcast or by direct transmission to any node that is waiting for the resource such that those nodes may interactively display download progress information to end users at those nodes. Alternatively, steps 216B and 216C may be combined into a single periodic packet transmission in which each packet is a “downloading” message that includes progress information.

Service-Enabled Product Updating Application

One exemplary implementation of the peering service described herein is a product updating service implementation and a service-enabled application having a shared agent. The agent is shared by an anti-virus application and a firewall application. The peering service is encapsulated in a single DLL that contains components for performing an update service, namely, a peering server having an HTTP server, a peering client, and a product updating service.

The product updating service determines what updates, if any, to request. If the product updating service determines that an update is necessary, the service client broadcasts an "I need" packet to request a specific URL for the necessary product updates. In other words, the peering service provides a mechanism for keeping service-enabled application, its engine, and its virus signature files up-to-date.

In particular, when a first computer or node boots, its product updater broadcasts an "I need" packet requesting for myupdate.cab file at a specified URL. The myupdate.cab file, e.g., approximately 7-8k in size, contains a script with instructions on how to check the current version of the product, engine, and virus signature files against the latest available version so that the product updater can determine if an update is necessary. This file may not be cacheable, so the service servers may not be able to offer it and can instead be obtained directly via the Internet.

If the product updating service determines, based on the myupdate.cab file, that an update is necessary, the product updating service, via the peering service, broadcasts an "I need" packet over the network. An update may include engine, DAT, and/or product updates. For any update files that are downloaded, whether directly from the Internet and/or from one or more of the peers on the network, the product updating service preferably checks to ensure that the updates have been digitally signed. Once the updates are authenticated, they are installed at the requesting node.

The product update service checks for updates at any suitable pre-defined intervals and/or upon occurrence of various events. For example, the product update service may check for updates upon boot or 5 minutes after boot, 6 hours after each

unsuccessful check, and/or upon a scheduled basis such as once a day, once every 12 hours after each successful check.

An update can include virus signature files (DATs), engine, and/or product update. DATs are typically updated weekly, such on a particular day of the week and are approximately 900-950k in size on average. The engine is usually updated every 2 to 3 months and is approximately 550-600k in size on average. The product is updated as hotfixes become available, typically every 6-8 weeks, or as new versions become available, typically every 4-6 months, and is approximately 700-750k in size on average.

In the current example, a complete update, including engine, virus signature files and product, comprises of six *.cab files, totaling approximately 2.25M. The six *.cab files for an update and their respective average sizes are listed below:

Myavdat.YYMMDDHHMM.cab	average 910k
Myxtrdat.YYMMDDHHMM.cab	average 16k
Mycioagt.YYMMDDHHMM.cab	average 370k
Vsasap.YYMMDDHHMM.cab	average 360k
Vseng9x.YYMMDDHHMM.cab	average 240k
Vsengine.YYMMDDHHMM.cab	average 340k

As any number of these *.cab files may need to be updated, each file is preferably requested via the peering service in a separate transaction. Thus, some or all of the needed *.cab file may be pulled from different nodes and/or the Internet.

FIG. 8 is a flowchart illustrating an exemplary secure product updating process implemented by a service-enabled product updating application over a peer-to-peer network. At step 242, the service-enabled product updating application retrieves a

catalog containing a listing of files for a current version of the product. At step 244, the service-enabled product updating application compares information contained in the catalog with versions currently installed at the node to determine if an update is needed. If a newer version is available, then the service-enabled product updating application
5 retrieves the newer version at step 246. Otherwise, the process 240 terminates.

Preferably, the newer version is located at a unique URL that is determined by the specific version to enables update to request a specific URL different from the URL the application used last time when it installed the currently installed version. For example, the unique URL may be generated by including an encoding of a timestamp in the file
10 name such that a file named VsASaP.CAB would named VsASaP.200101131511.CAB if it were released on January 13, 2001 at 3:11 PM.

The unique naming of URLs is desirable because the service-enabled product updating application attempts to locate local copies of specific URLs. Thus, the updating application may not be able differentiate between a cached copy of an old VsASaP.CAB
15 file and a copy of a most recent version of the VsASaP.CAB file without the additional version information encoded into the file name. With this unique file name, the updating application can ask the service to look for a copy of VsASaP.200101131511.CAB on the local network segment with assurance that it will obtain the desired file as listed in the update catalog.

20 The product updating application preferably verifies the integrity of the retrieved data at step 248. Such integrity verification is highly desirable particularly where the retrieved data may be actual executable binaries. Moreover, with data packets containing potentially critical data such as anti-virus product updates and signature files being

passed around in a network, security is of great concern. In addition, peer-to-peer distribution is more susceptible to attacks since the distributor of the data files does not have control of the peer servers that are passing the files around. Thus, the service-enabled updating application preferably implements digital signature and version information security measures at step 248 to ensure that the update is not compromised and are versioned properly.

In particular, step 248 may include verifying that contents of a retrieved data file signed with a digital signature or certificate have not been altered with use of digital signature security measure. Specifically, the digital signature is authenticated to verify that the file contents have not been altered after it was downloaded from the original source, i.e., verify the authenticity of the file. Any suitable digital signature authentication tool such as the 1024-bit VeriSign certificate may be utilized.

As another desired security measure, step 248 may additionally or alternatively include a version information security measure to guard against a retrieved data file that has been renamed from successfully masquerading as the specified file. Because the file name is utilized to indicate the version of the file, an old file may have been renamed as the requested file. The compromised file may be signed and unaltered and yet its contents would potentially be outdated or incompatible with the current version. Thus, a version information security measure is preferred.

According to a preferred embodiment, the version information security may be implemented by encoding an additional block of data containing version information within the digitally signed file such that the additional encoded data can be utilized to verify the file is indeed the desired file. In the case of product updating, a cabininfo.ini file

is optionally added to the MyUpdate.cab to specify the original file name for the package.

Thus, if an older file on the service server was renamed to match the current update filename, the embedded file description in that file would not match the one included in the MyUpdate.cab file on the client and the update file would be rejected.

5 Finally, at step 250, the product updating application installs the update package at the node.

As illustrated in the description above, the peering service facilitates in reducing or minimizing the number of service clients that have to obtain files or other resources
10 such as product update files via the Internet by using secure, peer-to-peer communication to distribute the files among client machines on a network, such as a LAN, via an intranet. The peering service enables secure, automatic distribution of the update files between service clients, independent of a network administrator or end-user, to keep the anti-virus and firewall application/service up-to-date with minimal impact to network
15 bandwidth.

Often, many computers on a network do not have the most up-to-date anti-virus and/or firewall files. Using the secure peering service allows for automatic and secure updating of such files and also reduces or eliminates the need for a network administrator to script anti-virus file updates. Furthermore, by efficiently spreading load and utilizing
20 resources across a local network over a high-speed LAN, a bandwidth crunch resulting from the computers pulling update files from the Internet is largely reduced. Thus, the peering service allows for ease of distribution of product upgrades and updates with a

minimal number of computers requiring to connect to the Internet to obtain the necessary files resulting in a reduced usage of Internet bandwidth.

The peering service also allows a given client to pull the data files from any node on the network, rather than having to connect to a centralized server that might require several additional network hops, resulting in an optimal use of network bandwidth to distribute updates.

FIGS. 9 and 10 illustrate a schematic and a block diagram, respectively, of an example of a general purpose computer system 1000 suitable for executing software programs that implement the methods and processes described herein. The architecture and configuration of the computer system 1000 shown and described herein are merely illustrative and other computer system architectures and configurations may also be utilized.

The illustrative computer system 1000 includes a display 1003, a screen 1005, a cabinet 1007, a keyboard 1009, and a mouse 1011. The mouse 1011 can have one or more buttons for interacting with a GUI (graphical user interface) that may be displayed on the screen 1005. The cabinet 1007 typically house one or more drives to read a computer readable storage medium 1015, system memory 1053, and a hard drive 1055, any combination of which can be utilized to store and/or retrieve software programs incorporating computer codes that implement the methods and processes described herein and/or data for use with the software programs, for example. Examples of computer or program code include machine code, as produced, for example, by a compiler, or files containing higher level code that may be executed using an interpreter.

Computer readable media may store program code for performing various computer-implemented operations and may be encompassed as computer storage products. Although a CD-ROM and a floppy disk 1015 are shown as exemplary computer readable storage media readable by a corresponding CD-ROM or floppy disk drive 1013, any other combination of computer readable storage media can be utilized. Computer readable medium typically refers to any data storage device that can store data readable by a computer system. Examples of computer readable storage media include tape, flash memory, system memory, and hard drive may alternatively or additionally be utilized. Computer readable storage media may be categorized as magnetic media such as hard disks, floppy disks, and magnetic tape; optical media such as CD-ROM disks; magneto-optical media such as floptical disks; and specially configured hardware devices such as application-specific integrated circuits (ASICs), programmable logic devices (PLDs), and ROM and RAM devices. Further, computer readable storage medium may also encompass data signals embodied in a carrier wave, such as the data signals embodied in a carrier wave carried in a network. Such a network may be an intranet within a corporate or other environment, the Internet, or any network of a plurality of coupled computers such that the computer readable code may be stored and executed in a distributed fashion.

Computer system 1000 comprises various subsystems. The subsystems of the computer system 1000 may generally include a microprocessor 1051, system memory 1053, fixed storage 1055 (such as a hard drive), removable storage 1057 (such as a CD-ROM drive), display adapter 1059, sound card 1061, transducers 1063 (such as speakers and microphones), network interface 1065, and/or scanner interface 1067.

The microprocessor subsystem 1051 is also referred to as a CPU (central processing unit). The CPU 1051 can be implemented by a single-chip processor or by multiple processors. The CPU 1051 is a general purpose digital processor which controls the operation of the computer system 1000. Using instructions retrieved from memory, the CPU 1051 controls the reception and manipulation of input data as well as the output and display of data on output devices.

The network interface 1065 allows CPU 1051 to be coupled to another computer, computer network, or telecommunications network using a network connection. The CPU 1051 may receive and/or send information via the network interface 1065. Such information may include data objects, program instructions, output information destined to another network. An interface card or similar device and appropriate software implemented by CPU 1051 can be used to connect the computer system 1000 to an external network and transfer data according to standard protocols. In other words, methods and processes described herein may be executed solely upon CPU 1051 and/or may be performed across a network such as the Internet, intranet networks, or LANs (local area networks), in conjunction with a remote CPU that shares a portion of the processing. Additional mass storage devices (not shown) may also be connected to CPU 1051 via the network interface 1065.

The subsystems described herein are merely illustrative of the subsystems of a typical computer system and any other suitable combination of subsystems may be implemented and utilized. For example, another computer system may also include a cache memory and/or additional processors 1051, such as in a multi-processor computer system.

The computer system 1000 also includes a system bus 1069. However, the specific buses shown are merely illustrative of any interconnection scheme serving to link the various subsystems. For example, a local bus can be utilized to connect the central processor to the system memory and display adapter.

5

While the preferred embodiments of the present invention are described and illustrated herein, it will be appreciated that they are merely illustrative and that modifications can be made to these embodiments without departing from the spirit and scope of the invention. Thus, the invention is intended to be defined only in terms of the following claims.

10

following claims.